MRTUCC 8 CPU

ARCHITETTURA A 8 BIT

DI FRANCESCO MARTUCCI, PER ESAME DI CALCOLATORI ELETTR. DEL 22/2/2019

INDICE

INTRODUZIONE

Ho realizzato questa CPU a 8 bit con l'intento di presentarla alla prova orale di Calcolatori elettronici, come personale approfondimento riguardo ad alcune parti del programma affrontato durante il corso, in particolare la «microarchitettura» e «livello RTL».

Progettare e disegnare in Logisim tale CPU (la prima nella mia carriera) è stata un'ardua sfida, ma mi ha permesso di comprendere molto meglio tutto ciò che ho studiato sui libri.

Inoltre ho potuto sperimentare direttamente cosa volesse dire dover progettare un sistema complesso, arrivando ad intuire cosa ci si aspetterà da me, quando sarò ufficialmente ingegnere.

CANALI IN/OUT (1 /2)

→Con canali s'intende quali segnali il circuito logico della CPU riceve/emette.

→Canali in:

- Reset_button : vengono resettati tutti i registri presenti nella CPU
- On_button : necessario per abilitare la memoria istruzioni e la CPU
- Main_clock : clock principale
- Clock_2 : clock secondario
- Instruction : istruzione da eseguire, prelevata dalla memoria istruzioni
- Data_in : dati caricati o dalla memoria dati o dalla periferica di input

CANALI IN/OUT (2/2)

→Canali out:

- Addr_instr (16 bit) : indirizzo dal quale caricare l'istruzione da caricare ed eseguire dalla memoria istruzioni
- Addr_data (16 bit) : indirizzo al quale accedere in memoria dati
- Data_memory_out (8 bit) : dato da scrivere in memoria dati
- Data_memory_enable : chip select della memoria dati
- Data_in_ctrl : se asserito ad 1, allora data_in sarà caricato con il valore inviato dalla periferica di input; se a 0, sarà caricato dalla memoria dati
- i_o_enable : chip select delle periferiche
- i_o_op : se asserito ad 1, allora la cpu sta richiedendo operazione di input, altrimenti sta richiedendo operazione di output
- Data_to_out (8 bit): dati inviati alla periferica di output

INSTRUCTION SET ARCHITECTURE

Questa CPU si ispira ai principi di progettazione delle architetture RISC

→Il parallelismo interno è di 8 bit

- La CPU è in grado di manipolare dati interi (da 0 a 255) con/senza segno

→Le istruzioni hanno dimensione fissa di 16 bit e sono raggruppate in 7 formati:

- aritmetico logico : add, sub, and, or, shl, shr, cmp
- Branch (salto) : bie, bilu, bils, jmp
- Accesso memoria dati : ld, st
- In/out : in, out
- Put
- Mov
- Nop

ARCHITETTURA MEMORIA INTERNA

→Nella CPU sono presenti i registri :

- Program Counter (16 bit) : il suo valore è l'indirizzo utilizzato per l'accesso alla memoria istruzioni
- 8 registri del Register File, ognuno da 8 bit (rappresentati dai numeri da 0 a 7)
- 1 registro dei flags da 4 bit

→L'implementazione dell' architettura della memoria interna è realizzata con d-flip-flop organizzati in registri da 8 bit (register file) e 4 bit (flags)

TABELLA DEI CODICI OPERATIVI

nop	0	0	0	0	0
add	0	0	0	0	1
sub	0	0	0	1	0
and	0	0	0	1	1
or	0	0	1	0	0
shl	0	0	1	0	1
shr	0	0	1	1	0
cmp	0	0	1	1	1
bie	0	1	0	0	0
bilu	0	1	0	0	1
bils	0	1	0	1	0
jmp	0	1	0	1	1
_	_	_	_	_	_
_	_	_	_	_	_
mov	0	1	1	1	0
put	0	1	1	1	1
ld	1	0	0	0	0
st	1	0	0	0	1
out	1	0	0	1	0
in	1	0	0	1	1

Ogni formato è accomunato dai primi 5 bit, i quali compongono il codice operativo, che identifica l'istruzione da eseguire.

Ogni configurazione non indicata in tale tabella è considerata vietata e se caricata nella CPU scatterà un'interruzione e verrà bloccata l'esecuzione (vedi INTERRUZIONI).

ISTRUZIONI ARITMETICO LOGICHE (1/2)

op_code + reg_dst + reg_src1 + reg_src2

5 bit + 3 bit + 3 bit + 3 bit + 2 bit di indifferenze

→Gli operandi utilizzati saranno il contenuto dei registri source, mentre il risultato sarà posto nel registro destinazione.

→Add (addition) esegue l'addizione tra gli operandi.

→Sub (substraction) esegue la sottrazione tra gli operandi.

ISTRUZIONI ARITMETICO LOGICHE (2/2)

→And e Or eseguono, bit a bit, l'operazione di logica booleana di and e or

→Shl e Shr eseguono lo shift rispettivamente verso sinistra e verso destra, del valore di src1 di quanto indicato da src2 riempiendo eventualmente con bit zero (si assume che livelli superiori garantiscano che in tali istruzioni reg_src2 non contenga un valore superiore ad 8, altrimenti si incorrerà in un Undefined Behaviour).

→Cmp (compare) è eseguita come fosse una sub, così da impostare i bit di flag, anche se la Control Unit blocca la scrittura nel registro destinazione del risultato

Per ulteriori dettagli vedi ALU

REGISTRO DEI FLAGS

→Registro a 4 bit che memorizza lo stato della parola in uscita dalla ALU:

- Zero_Flag : attivo (=1) se il risultato in uscita dalla ALU è effettivamente 0.
- Sign_Flag : indica il segno del risultato in uscita dalla ALU; cioè è uguale al suo bit più significativo.
- Carry_Flag : se attivo, indica che vi è stato un riporto o un prestito sul bit più significativo del risultato di ALU.
- Overflow_Flag : se attivo, indica che vi è stato un overflow, cioè il risultato non è rappresentabile correttamente con i bit a disposizione. Si verifica quando la somma tra due numeri positivi risulta un numero negativo o quando la somma tra due numeri negativi risulta un numero positivo.

→Tale registro è utilizzato per verificare se il salto va preso e per eventuali interruzioni (vedi meglio EXCEPTION UNIT)

ISTRUZIONI DI SALTO (1/2)

Op_code + branch_offset

5 bit + 11 bit

→Per tutte le istruzioni di salto, l'indirizzo dell'istruzione a cui trasferire il flusso di controllo è costruito utilizzando la formula: Program_counter + branch_offset

Inoltre si assume che branch_offset sia rappresentato in complemento a 2, permettendo salti a indirizzi minori o maggiori nel PC attuale, permettendo salti relativi da 0 a +/- 2048.

Eseguire dei «long branch» superiori a tale soglia è possibile concatenando opportunamente dei branch. Si assume che tale gestione avvenga ai livelli superiori.

ISTRUZIONI DI SALTO (2/2)

→Le istruzioni di salto utilizzano i bit di flag, quindi è necessario che siano sempre precedute dall'operazione Cmp per poter ottenere un risultato coerente.

→Bie (branch if equal) : il salto viene eseguito se il bit Zero_Flag = 1

→Bilu (branch if lower unsigned) : il salto viene eseguito se il primo operando di Cmp è minore del secondo, considerando tali operandi unsigned; cioè se Carry_Flag = 1.

→Bils (branch if lower unsigned) : il salto viene eseguito se il primo operando di Cmp è minore del secondo, considerando tali operandi signed; cioè se Sign_Flag diverso da Overflow_Flag

→Jmp (jump) : salto incondizionato.

Per ulteriori dettagli vedi UPDATE PC UNIT

ACCESSO MEMORIA DATI

Per ulteriori dettagli vedi DATA MEMORY

Op_code + reg_dst + low_address

5 bit + 3 bit + 8 bit

→L'indirizzamento della memoria dati è indiretto e vi è allineamento al byte, l'indirizzo a cui accedere è costituito «logicamente» da due parti:

- High_part : è il valore contenuto nel registro n.7
- Low_part : è il campo contenuto direttamente nell'istruzione

Quindi si assume che quando viene caricata un'istruzione di accesso alla memoria dati, il registro n.7 sia stato opportunamente preparato.

→Ld (load) : carica il valore presente nella memoria dati, all'indirizzo costruito, in reg_dst.

→St (store) : scrive in memoria dati, all'indirizzo costruito, il valore presente in reg_dst.

ACCESSO PERIFERICHE I/O

Op_code + reg_dst

5 bit + 3 bit + 8 bit di indifferenza

→Vi è un canale out dedicato allo scambio di dati con una periferica, mentre un canale in, il quale è in concorrenza tra data memory e periferica_in. Vi sono istruzioni dedicate per I/O.

→In : il valore sul canale_in viene scritto su reg_dst

→Out : il valore sul canale_out viene inviato alla periferica_out

Per ulteriori dettagli vedi I/O control Unit

ISTRUZIONI SPECIALI

→Put : op_code + reg_dst + const_value

scrive il valore const_value in reg_dst

→Mov : op_code + reg_dst + reg_src1

scrive il valore presente in reg_sr1 su reg_dst

→Nop (no operation) : op_code

la CPU non esegue alcuna operazione, se non l'incremento del PC

INTERRUZIONI

→Vi sono due tipi di interruzioni:

- Richiesta dall'esterno tramite reset_button
- Eccezione interna (trasparente a livelli superiori)

→Tramite reset_button si resettano tutti i registri interni alla CPU tranne il Program Counter, che viene impostato al valore 0x0001; quindi dopo l'interruzione riprenderà l'esecuzione dall'istruzione caricata dall'indirizzo su detto della memoria istruzioni.

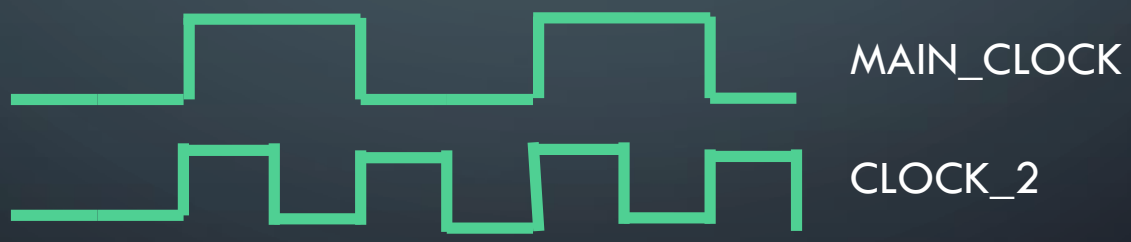
→Se in un qualunque istante l'Exception_Unit rileva un errore, Il Program Counter viene impostato al valore 0x0000. Con errore si può intendere:

- Overflow (nelle istruzioni aritmetico logiche e del program counter)
- Op_code non valido caricato nella cpu

MICROARCHITETTURA

Questa è una cpu a singolo ciclo di clock, cioè l'istruzione viene interamente eseguita in un unico ciclo di clock.

→Sono presenti due clock (principale e secondario) per implementare un meccanismo di controllo della scrittura sul register file :



UPDATE PC UNIT

→Si occupa di aggiornare il Program Counter ad ogni clock : se l'istruzione da eseguire è di salto, calcola tale indirizzo (PC + BRANCH_OFFSET), altrimenti (PC+1).

→Segnali IN di dato:
- Program Counter
- Branch offset

→Segnali IN di controllo:
- Update_PC_unit
- Branch_ctrl : pilota un multiplexer per selezionare o PC+1 o PC+BRANCH_OFFSET

→Segnali OUT di dato:
- New_PC

→Segnali OUT di controllo:
- Pc_overflow

Per dettagli vedi UPDATE PC CONTROL UNIT

INSTRUCTION MANAGEMENT

→Non è una vera e propria rete logica, ma solo un'unità di smistamento dei bit che compongono l'istruzione, inviandoli alle opportune unità che compongono la CPU.

→Segnali IN:

- INSTRUCTION : l'istruzione prelevata dalla memoria istruzioni

→Segnali OUT:

- OP_CODE
- REG_DST
- REG_SRC1
- REG_SRC2
- BRANCH_OFFSET
- CONST_VALUE

N.B. Per i dettagli consultare lo schema logico

REGISTER FILE (1/4)

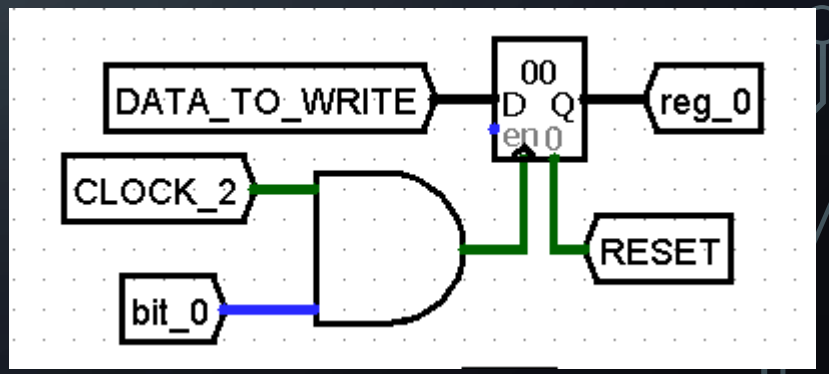
→Unità interna di memorizzazione temporanea dei dati

→Sono presenti 8 registri (numerati da 0 a 7)

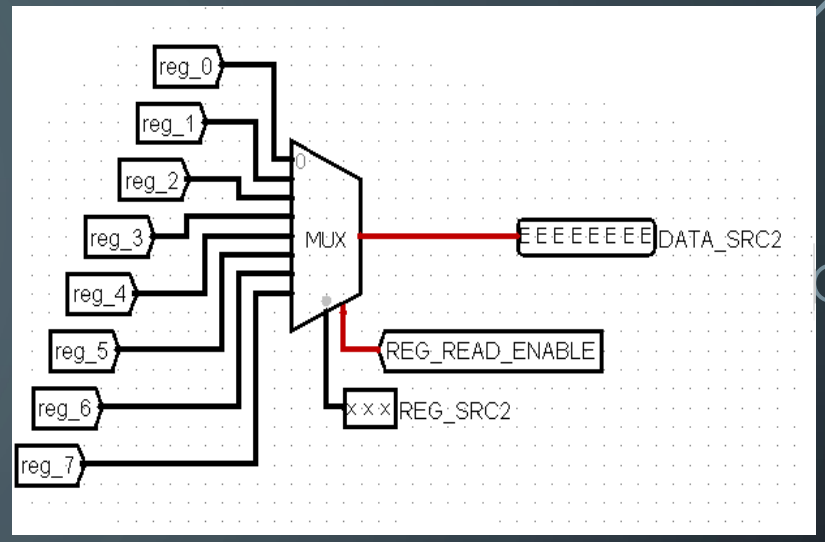
→Ogni registro presenta in ingresso un segnale di dato (8 bit), ciò che si vorrebbe scrivere su di esso, e un segnale di controllo che consente la scrittura solo se sono attivi il clock E la richiesta di scrittura dal segnale «bit_n», pilotato da un decoder (vedi seguito).

È anche presente il segnale di RESET il quale azzera il contenuto.

→In uscita è presente un segnale di dato (8 bit).



REGISTER FILE (2/4)



→Emette in output tre segnali di dati (8 bit):

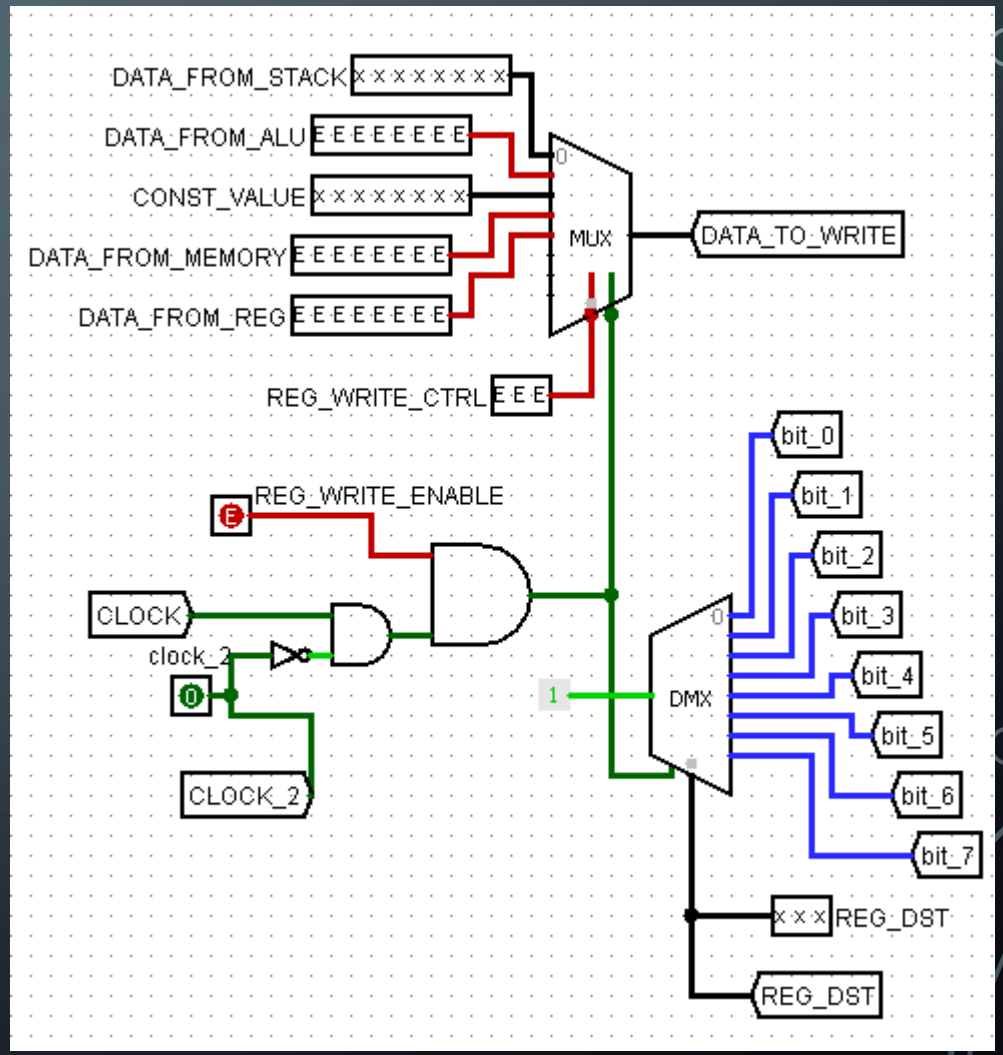
- I primi due sono REG_SRC1/REG_DST e REG_SRC2 che saranno gli operandi della ALU; sono prelevati da due degli otto registri attraverso multiplexer pilotati dalla Control Unit in funzione dell'istruzione caricata, i quali prevedono segnale di abilitazione.
- REG_7 il quale viene inviato alla data_memory_management_unit.

REGISTER FILE (3/4)

→I dati scritti sul REG FILE possono provenire da: ALU, data memory, da un altro registro del register file, da const_value (campo dell'istruzione).

→Il segnale, tra i su detti, utlizzato per eseguire l'operazione è filtrato da un multiplexer pilotato dalla Control Unit.

→Inoltre un decoder, pilotato dalla Control Unit abilita la scrittura del segnale prelevato dal multiplexer su un determinato registro.



REGISTER FILE (4/4)

→Segnali IN di dato:

- Data_src1 (nel caso di op. mov)
- Data_from_memory
- Const_value
- Data_from_ALU

→Segnali OUT:

- Reg_7
- Data_src1
- Data_src2

→Segnali IN di controllo:

- Write_enable
- Read_enable
- Reg_Write_ctrl : segnale di controllo del multiplexer di selezione del dato da scrivere
- Reg_dst
- Reg_src1
- Reg_src2
- Data_mem_op : segnale di controllo del multiplexer di selezione dell'out Data_src1.
- Reset, main_clock, clock_2

ARITHMETIC LOGIC UNIT (1 /2)

→Si occupa dell'elaborazione aritmetico-logica dei dati, i quali sono prelevati solo dal register file ed ogni risultato è inviato solo al register file.

→Segnali IN di dato:

- Data_src1
- Data_src2

→Segnali IN di controllo:

- Alu_enable
- Alu_op_ctrl : segnale di controllo del multiplexer che preleva il risultato di OUT

→Segnali OUT di dato:

- Alu_result

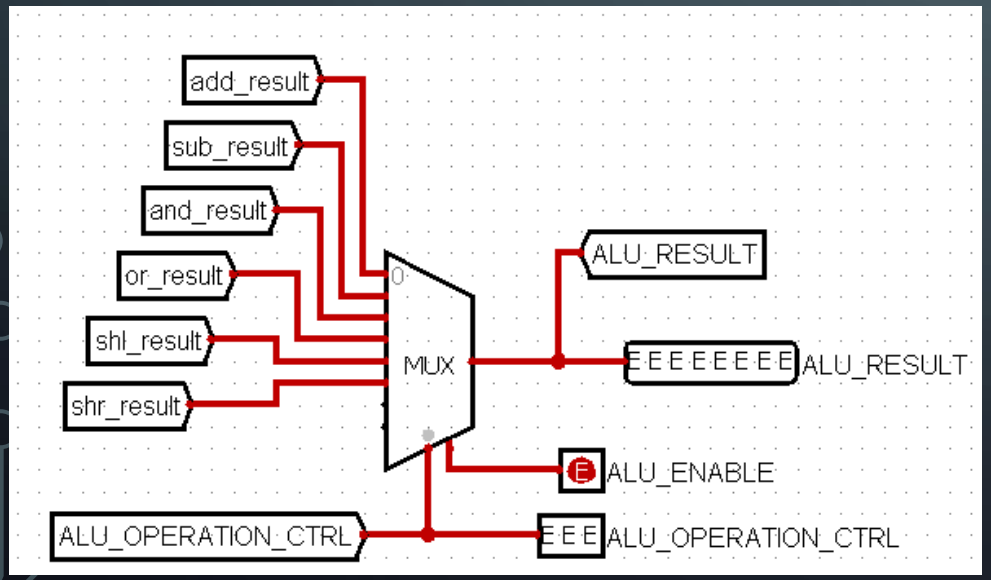
→Segnali OUT di dato:

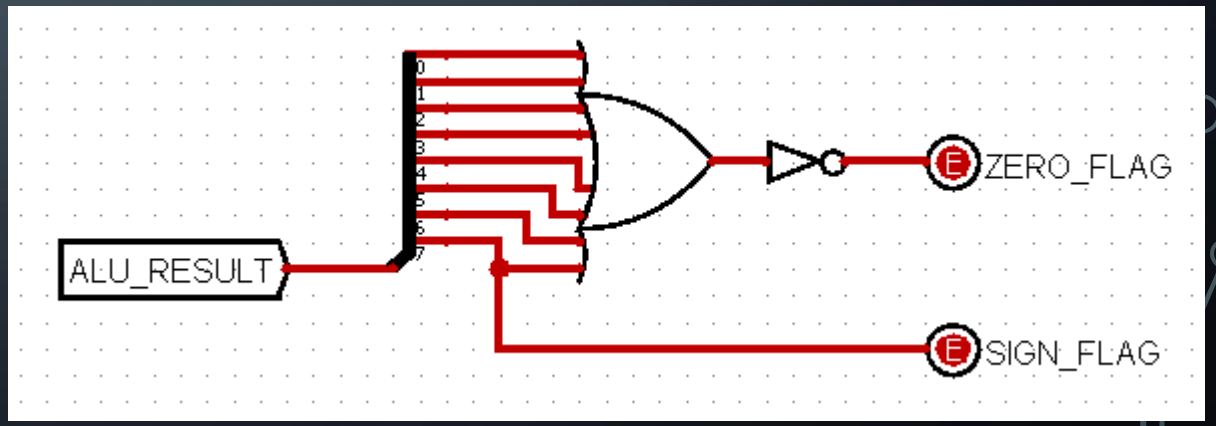
Registro dei FLAGS

ARITHMETIC LOGIC UNIT (2/2)

→Ogni operazione alu prevista dall'ISA viene eseguita in parallelo e ogni risultato confluisce in un multiplexer, il quale ne preleva uno solo, in base ai segnali di controllo ricevuti dalla Control Unit.

→Inoltre vengono generati i bit di FLAGS.





DATA MEMORY MANAGEMENT

→Si occupa di costruire l'indirizzo per poter accedere alla memoria dati nel caso di operazioni di accesso ad essa. L'indirizzo viene costruito «unendo» i valori di Reg_7 (bit più significativi) e del campo dell'istruzione Const_value (bit meno significativi).

→Segnali IN di dato:

- Data_src1
- Reg_7
- Const_value

→Segnali IN di controllo:

- Main_clock

→Segnali OUT di dato:

- Data_mem_addr (16 bit)
- Data_mem_Data

EXCEPTION UNIT

→Sotto-unità della Control Unit che si occupa di inviare un apposito segnale di interruzione nel caso rilevi degli errori.

→Con errore si può intendere:

- Overflow (nelle istruzioni aritmetico logiche e del program counter)
- Op_code non valido caricato nella cpu

→Alla rilevazione dell'errore, la Control Unit interromperà l'esecuzione dell'istruzione caricata (disabilitando tutte le unità con il segnale «exception») e imposterà il PC al valore 0x0000h, assumendo che il valore contenuto a tale indirizzo sia un puntatore ad una routine di servizio.

EXCEPTION UNIT (2/2)

→Segnali IN di dato:

- Op_code (5 bit)
- Overflow_flag
- Pc_overflow

→Segnali IN di controllo:

- Main_clock
- Clock_2

→Segnali OUT:

- exception

INSTRUCTION MEMORY

→La cpu comunica con l'instruction memory attraverso un bus degli indirizzi a 16 bit e un bus dei dati a 16 bit (dimensione dell'istruzione); in altre parole la memoria è vista come un blocco logico avente parola 16 bit.

→La cpu emette segnale di chip select e di enable, ma nel progetto in Logisim si sfrutta solo il segnale di clock e on_button per far emettere i dati alla memoria.

DATA MEMORY

→La cpu comunica con la memoria dati attraverso:

- Bus degli indirizzi a 16 bit
- Bus dei dati a 8 bit (per la scrittura in memoria)
- Bus dei dati a 8 bit (per la lettura da memoria) multiplexato assieme al bus dei dati della periferica di input

→è controllata dalla cpu attraverso i segnali di chip_select, data_l_w (se 0 lettura, altrimenti scrittura), oltre al segnale di clock.

CONTROL UNIT

→Si occupa di inviare i necessari segnali di controllo ad ogni singolo componente esposto fino ad ora, per garantire il corretto funzionamento della cpu.

→Essendo questa cpu a singolo ciclo di clock, l'unità di controllo è sostanzialmente una rete combinatoria (anche se qui risiedono i registri di FLAGS).

→è stata internamente organizzata in modo tale che ogni sotto unità si occupi dei segnali di controllo necessari ad una particolare unità del data path, oltre ad una per I/O e una (già esposta) per le eccezioni.

ALU CONTROL UNIT

→Si occupa di pilotare la ALU, in base all'op_code caricato, emettendo segnale di enable e il segnale di operation_ctrl, il quale controllerà il multiplexer delle operazioni della alu.

→Segnali in:

- Op_code (5bit)
- Exception

→Segnali out:

- Alu_operation_ctrl
- Alu_enable

I/O CONTROL UNIT

→Si occupa di pilotare le periferiche di i/o e il bus di entrata alla cpu data_in.

→Segnali in:

- Op_code (5bit)
- Exception

→Segnali out:

- i_o_enable
- i_o_op : se 0 operazione di input, altrimenti operazione di output
- Data_in_ctrl : se 0 sul bus data_in passeranno i dati provenienti dalla memoria dati, altrimenti i dati provenienti dalla periferica di input

UPDATE PC CONTROL UNIT

→Si occupa di pilotare l'unità di aggiornamento del PC.

→Segnali in:

- Op_code (5 bit)
- Exception
- Flags (5 bit)

→Segnali out:

- Upc_enable
- Branch_ctrl : se 0 l'unità UPC emette il PC incrementato di 1, altrimenti il PC incrementato con il dovuto indirizzo spiazzamento.

INSTR./DATA MEMEORY CONTROL UNIT

→Si occupa di pilotare la memoria istruzioni, emettendo il segnale instr_mem_enable, in base all'op_code caricato e al segnale exception.

→Si occupa di pilotare la memoria dati.

→Segnali in:

- Op_code (5 bit)
- exception

→Segnali out:

- Data_mem_enable
- Data_mem_l_w : se 0 operazione di lettura dalla memoria, altrimenti operazione di scrittura

REGISTER FILE CONTROL UNIT

→Si occupa di pilotare il register file

→Segnali in:

- Op_code (5 bit)
- Exception

→Segnali out:

- Reg_w_enable : abilita la scrittura dei registri
- Reg_l_enable : abilita la lettura dei registri
- Reg_w_ctrl (3 bit) : pilota il multiplexer che seleziona la sorgente del dato da scrivere sui registri
- Data_mem_op : abilita un multiplexer che permette al bus dati della memoria dati di leggere reg_dst.